6.0 SOFTWARE QUALITY ASSESSMENT

The purpose of the analysis described in this report is to support the verification of ESAMS by analyzing the structure of its source code from a software engineering perspective. No attempt was made in this analysis to analyze or evaluate the output of ESAMS.

6.1 MODEL DESCRIPTION

As with any software system with an active Product Improvement Program (PIP), ESAMS has been released numerous times in many versions. The information in this section describes the specific version that was the subject of this analysis. Table 6.0-1 lists the pertinent release information of the version analyzed.

TABLE 6.0-1. ESAMS Release Specifics.

NAME:	Enhanced Surface-to-Air Missile Simulation
ACRONYM:	ESAMS
VERSION:	2.7
DATE OF RELEASE:	November 1995

6.1.1 Functional Description

The program simulates an encounter between surface-to-air missile (SAM) system and a single intruding aircraft. The primary model result is probability of target kill (P_k); however ESAMS can examine other areas such as missile flight path, guidance characteristics, and the effect of electronic countermeasures (ECM) and terrain on an engagement. ESAMS simulates sensor lock-on, tracking, missile aerodynamic propulsion, guidance and control. It computes P_k, missile distance and missile flight time.

6.1.2 Source Code Statistics

Appendix D contains a listing of source code files for the main programs, subroutines, and functions that comprise ESAMS 2.7, ESAMS 2.7β and ESAMS 2.6.3β. File changes are summarized in Tables 6.0-2 and 6.0-3.

TABLE 6.0-2. Source Code Statistics ESAMS 2.6.3β vs. ESAMS 2.7β.

	NUMBER OF FILES	PERCENT
ESAMS 2.6.3β	641	
ESAMS 2.7β	656	
no change	324	49.4%
major change [1]	185	28.2%
minor change [2]	67	10.2%
comments only	7	1.1%
new files	73	11.1%
files not used from ESAMS 2.6.3β	58	

TABLE 6.0-3. Source Code Statistics ESAMS 2.7β vs. ESAMS 2.7.

	NUMBER OF FILES	NUMBER OF BYTES	PERCENT
ESAMS 2.7β	656	3,541,187	
ESAMS 2.7	656	8,356,090	236% larger
no change (except classification)	613		93.4%
major change [1]	20		3.0%
minor change [2]	18		2.8%
comments only	5		.8%
Classification			
Unclassified	425		65%
Secret/NOFORN	231		35%

[1] A major change is defined as one that affects execution.
[2] A minor change is one made for aesthetic reasons.

6.2 SOFTWARE QUALITY ASSESSMENT APPROACH

ESAMS 2.7 is a large program, consisting of two main program modules and over 700 subroutines, functions, include files and other forms of subordinate modules. One of the main programs is contained in file PREPRO.FOR, a pre-processor as the name implies. The other main program is contained in file ZINGER.FOR, the main program for ESAMS. Unlike the main ESAMS program, PREPRO.FOR is largely self-contained with the main program and eighteen (18) associated subroutines contained in the same file. The main ESAMS program (ZINGER.FOR), on the other hand, is very modular with called subroutines contained in individual files.

As a result of the size of ESAMS, and time and resource limitations, the analysis focused on a sample set of 100 subroutines as listed in Appendix E. The results from this sample were then augmented by analysis of another 100 modules listed in Appendix F. Even

though this sample represents about one-third of the code, it should provide a reasonable indication of the construction of the simulation as a whole.

The subjective nature of quality evaluations combined with personal preferences can result in different assessments of the same code from different analysts. The balance of this section describes the four measure of effectiveness (MOEs) chosen for the SQA and attempts to define an appropriate set of contributing factors (criteria) for each MOE. Clear descriptions of evaluation criteria and consistency of rating application by evaluators is essential to a fair assessment of quality.

6.2.1 MOE #1 Use of Standards

6.2.1.1 Readability

Programmer style varies, but there should be certain consistencies from routine to routine and they should be organized in a similar order. There should be a header, version number, purpose, variable declaration, include statements, all with appropriate comments in a format that is pleasing to the eye. Code should be neat and clear with appropriate comments and line spacing as the programmer sees fit to make the code easy to read.

6.2.1.2 Modifiability

A well designed, modular structure that adheres to good software development practices, with meaningful variable names and descriptive comments will allow a model to be easily modified, updated or expanded to include additional features or capabilities. A high score in modifiability is dependent on high scores in other criteria (e.g., readability, variable declarations, naming conventions).

6.2.1.3 ANSI standards

Each module analyzed was compared to the American National Standard FORTRAN-77 X3.9-1978 standard for compliance. This standard specifies the form and establishes the interpretation of programs expressed in the FORTRAN language. The purpose of this standard is to promote portability of FORTRAN programs for use on a variety of data processing systems. The requirements, prohibitions, and options specified in this standard, generally refer to permissible forms and relationships for standard conforming programs.

6.2.2 MOE #2 Programming Conventions

6.2.2.1 Use of Comments and Headers

Comments should consist of large descriptive headers containing the version number of the source code complete with authors name and dates that indicate when the code was written with a revision history. This is not required but recommended. The header should also contain a technical description defining the routines purpose and functionality. Variable names, local and global should be listed, one to a line with clear definitions. Each subroutine that is called should also be listed with a short description of its intent. Embedded comment should be plentiful, clear and adequately provide a good understanding of what is intended. Classification markings, top and bottom will also be considered.

6.2.2.2 Use of Formatted Statements

The source code should be formatted to aid readability and modifiability. Variable declarations, COMMON, TYPE and DATA statements should be located at the beginning of the module, FORMAT and other non-executable statements should be at the end of the module. Readability is improved if embedded comments are written in lower case with blank lines appropriately placed to highlight certain areas. DO loops and IF blocks need to be indented to improve readability and understandability.

6.2.2.3 Logical I/O Devices

All file I/O units should be initialized as variables with no reference to specific units by number. The variable names should be meaningful and properly commented to facilitate understanding.

6.2.2.4 Variable Declarations

The term variable is used to denote stored data items represented by symbolic names associated with a storage location. Variables are classified by data type. Type declaration statements explicitly define the data type of specified symbolic names. There are two forms of type declaration statements: numeric type declaration (byte, logical, integer, real, double precision, complex, and double complex) and character. Defaults will not be allowed, and declared variables without comments will be rated poorly. The modules should all contain the statement IMPLICIT NONE. The use of IMPLICIT NONE forces the programmer to define each and every one of the variables in use before it is used. As a result, no default variable types are accepted. This allows the code to be more easily understood and modified.

6.2.2.5 Variable Initialization

Initializations may be performed in many ways in FORTRAN. Assignment statements (e.g., VALUE=NUMBER), DATA statements, DIMENSION statements, and EQUIVALENCE statements may all be used; however, the use of the latter is generally considered poor practice. The expression NUMBER should be evaluated and verified that it conforms to the range requirements of VALUE. Comments, as always, will contribute to a higher rating.

6.2.2.6 Variable Naming Conventions

A well defined naming convention for variables should be established before coding begins. Consistent, meaningful variable names with comments aid in the readability and modifiability of a module.

6.2.2.7 Algorithm Clarity

Algorithms should be developed in a modular fashions. Long complicated equations should be broken up into smaller, readable well commented sections. Each element in the calculation should be executed on its own line if possible. No attempt was made to verify the correctness of the algorithms examined.

6.2.3 MOE #3 Computational Efficiency

6.2.3.1 Mixed Mode Calculations

Combining integers and real numbers in a calculation, although not prohibited, can result in errors due to the different internal representation of the two variable types if used incorrectly. In general this is considered a poor programming practice and should not be used without sufficient comments that describe why the author feels it is necessary in the computation.

6.2.3.2 Use of Library Functions

FORTRAN library function names are called intrinsic function names. Intrinsic functions perform frequently used mathematical computations. Intrinsic functions should be used in complex calculations rather than the programmer re-writing code that is available for free. This will avoid programming errors, keep the calculation shorter and is self documenting.

6.2.3.3 Nested Computations

DO and IF loops should be indented to improve readability. This is not required by the compiler, but several levels of DO or IF statements not properly aligned is extremely difficult to follow. Additionally, equal levels of complicated calculations should be lined-up vertically with lower case comments, which improves readability and modifiability.

6.2.4 MOE #4 Maintainability

6.2.4.1 Portability

The ability to execute code on several different platforms is desirable. Machine dependent algorithms and routines should be avoided.

6.2.4.2 Memory Management

In FORTRAN, there are many optimization techniques used by the compiler. One way is to reduce I/O system overhead which is controlled by how you set up I/O operations in the source code. For instance, an I/O list consisting of a single unformatted element does not have to be buffered in the Run-Time Library buffers. Also, implied DO loops consisting of a single unnested element are transmitted as a single call to the Run-Time library. To obtain minimum I/O processing, the record length of direct access sequential organization files should be a multiple of the device block size of 512 bytes. Also, memory space can be optimized by controlling data size and code size, dead variable elimination, dead code elimination and elimination of unreachable code.

6.2.4.3 Use of Common Blocks

One of the most effective controls available to the programmer is by the efficient use of COMMON blocks and arrays. The programmer must exercise common sense design techniques such as not defining an overly large array that isn't used, or declaring variables that are never referenced.

6.2.4.4 Modularity

Each subroutine or function should be short and to the point providing one specific purpose clearly defined and documented.

6.2.4.5 Subroutine Traceability

An important part of documentation is to list in the header other subroutines that are called by the routine you are evaluating, and what their function is. It is also important, when looking at a particular routine, to know what the calling routine was. Being able to understand the program flow from one routine to another, forward and backwards greatly improves modifiability. It is understood that this may be difficult at times since many routines may call the same subroutine, but some type of documentation is desirable.

6.3 RESULTS AND CONCLUSIONS

6.3.1 Summary

The static analysis conducted here lends itself to compiling metrics and will provide valuable insight into how the code was developed and how easily others can understand the programmers intent, it provides limited information on how well it performs. A more dynamic evaluation should be conducted to fully evaluate and verify that the programmers intent is successful upon execution.

Each of the modules within the sample set was critically inspected according to each of the criteria listed beneath each MOE on the Software Evaluation Work Sheet. The compliance levels for each criterion across all the modules in the sample set were tallied, and the average was assigned to the performance level for that criteria.

6.3.2 Scoring Procedure

The average of each of the performance levels for each criteria is summarized in Tables 6.0-4 and 6.0-5 for the first group of 100 routines evaluated (Appendix E) and Tables 6.0-6 and 6.0-7 for the second group of 100 routines evaluated (Appendix F). A perfect score of 5.00 was awarded to 6 of the 18 categories in group 1, which means all 100 routines were rated excellent in those 6 categories. A rating between 4.00 and 4.99 was tabulated in the acceptable column and 3.99 and below in the poor practice column, of which there were none.

Table 6.0-6 shows how the scores were computed in Appendix E. The same procedure was used for Appendix F, which is summarized in Table 6.0-9. For example, if you take Criterion #1 Readability under MOE #1, 58 of the 100 routines were rated acceptable at 4 points each totaling 232 points, and 42 of the 100 routines were rated excellent at 5 points each totaling 210 points. The sum of 232 and 210 is 442 divided by 100 routines gives an average of 4.42.

The sum of the four (4) MOEs totals 87.33 of a possible 90 points. Since a maximum of 90 is an odd number to relate to, (100% is more appropriate), it is easier to look at 87.33 points being 97.03% of the 90 maximum possible points. Therefore, of the 100 modules evaluated, they are 97% in compliance with the chosen criteria.

6.3.3 Conclusions

The developers of ESAMS 2.7 adopted excellent programming conventions and deserve credit for their efforts. Large descriptive formatted headers with extensive comments were at the beginning of 98 of the 100 modules evaluated. The analysis indicated that all code adheres to the ANSI FORTRAN-77 standard, no mixed mode calculations were

encountered, library functions were used according to the criteria and all logical devices were assign meaningful names. COMMON blocks were used effectively as well as memory management. There were some GOTO statements found, but they were well documented and clearly understood. Ratings didn't suffer because of them. Most of the modules were 2-3 pages in length.

Much of the source code was formatted nicely, but additional line spacing would have greatly improved readability. Readability would also be improved if embedded comments were in lower case, with source code written in upper case with appropriate line spacing. Approximately half of the sample set was programmed this way. The lack of comments on variable names was evident in a minority of cases, which made the code difficult to follow when encountered.

All modules listed which subroutines they called, but only three of the one hundred analyzed listed the calling routine for the module being examined. It is understood that many routines may call the same module, but to improve modifiability one must know the path from one module to another, both forward and back.

TABLE 6.0-4. ESAMS 2.7 Performance Analysis Results.

Criterion	Performance Level		
	Poor	**Acceptable**	**Excellent**
MOE #1 - Use of Standards:		4.71	
Criterion #1: Readability		4.42	
Criterion #2: Modifiability		4.72	
Criterion #3: ANSI standards		4.98	
MOE #2 - Programming Conventions:		4.90	
Criterion #1: Use of comments and headers		4.80	
Criterion #2: Use of formatted statements		4.88	
Criterion #3: Logical I/O devices			5.00
Criterion #4: Variable declarations		4.84	
Criterion #5: Variable initialization		4.90	
Criterion #6: Variable naming conventions		4.94	
Criterion #7: Algorithm clarity		4.91	
MOE # 3 - Computational Efficiency:		4.99	
Criterion #1: Mixed mode calculations			5.00
Criterion #2: Use of library functions			5.00
Criterion #3: Nested computations		4.96	
MOE # 4 - Maintainability:		4.80	
Criterion #1: Portability			5.00
Criterion #2: Memory management			5.00
Criterion #3: Use of COMMON blocks			5.00
Criterion #4: Modularity		4.95	
Criterion #5: Subroutine traceability		4.03	

TABLE 6.0-5. ESAMS 2.7 Performance Summary.

MOE #1 - Use of Standards	4.71
MOE #2 - Programming Conventions	4.90
MOE #3 - Computational Efficiency	4.99
MOE #4 - Maintainability	4.80
Average per MOE	4.85
Total Score	87.33
Relative Performance	97%

TABLE 6.0-6. ESAMS 2.7 Performance Analysis Scoring Worksheet.

Criterion	**Performance Level**			
	Poor	**Acceptable**	**Excellent**	**Score**
MOE #1 - Use of Standards:				
Criterion #1: Readability		58*4=232	42*5=210	442/100=4.42
Criterion #2: Modifiability		28*4=112	72*5=360	472/100=4.72
Criterion #3: ANSI standards		2*4=8	98*5=490	498/100=4.98
MOE #2 - Programming Conventions:				
Criterion #1: Use of comments and headers	2*3=6	16*4=64	82*5=410	480/100=4.80
Criterion #2: Use of formatted statements		12*4=48	88*5=440	488/100=4.88
Criterion #3: Logical I/O devices			100*5=500	500/100=5.00
Criterion #4: Variable declarations	5*3=15	6*4=24	89*5=445	484/100=4.84
Criterion #5: Variable initialization	1*3=3	8*4=32	91*5=455	490/100=4.90
Criterion #6: Variable naming conventions		6*4=24	94*5=470	494/100=4.94
Criterion #7: Algorithm clarity		9*4=36	91*5=455	491/100=4.91
MOE # 3 - Computational Efficiency:				
Criterion #1: Mixed mode calculations			100*5=500	500/100=5.00
Criterion #2: Use of library functions			100*5=500	500/100=5.00
Criterion #3: Nested computations		4*4=16	96*5=480	496/100=4.96
MOE # 4 - Maintainability:				
Criterion #1: Portability			100*5=500	500/100=5.00
Criterion #2: Memory management			100*5=500	500/100=5.00
Criterion #3: Use of COMMON blocks			100*5=500	500/100=5.00
Criterion #4: Modularity		5*4=20	95*5=475	495/100=4.95
Criterion #5: Subroutine traceability		97*4=388	3*5=15	403/100=4.03

TABLE 6.0-7. ESAMS 2.7 Performance Analysis Worksheet.

Criterion	Performance Level		
	Poor	**Acceptable**	**Excellent**
MOE #1 - Use of Standards:		4.79	
Criterion #1: Readability		4.67	
Criterion #2: Modifiability		4.72	
Criterion #3: ANSI standards		4.99	
MOE #2 - Programming Conventions:		4.88	
Criterion #1: Use of comments and headers		4.65	
Criterion #2: Use of formatted statements		4.94	
Criterion #3: Logical I/O devices			5.00
Criterion #4: Variable declarations		4.80	
Criterion #5: Variable initialization		4.89	
Criterion #6: Variable naming conventions		4.90	
Criterion #7: Algorithm clarity		4.95	
MOE # 3 - Computational Efficiency:		4.99	
Criterion #1: Mixed mode calculations			5.00
Criterion #2: Use of library functions			5.00
Criterion #3: Nested computations		4.97	
MOE # 4 - Maintainability:		4.77	
Criterion #1: Portability			5.00
Criterion #2: Memory management			5.00
Criterion #3: Use of COMMON blocks			5.00
Criterion #4: Modularity		4.86	
Criterion #5: Subroutine traceability		4.00	

TABLE 6.0-8. ESAMS 2.7 Performance Summary.

MOE #1 - Use of Standards	4.79
MOE #2 - Programming Conventions	4.88
MOE #3 - Computational Efficiency	4.99
MOE #4 - Maintainability	4.77
Average per MOE	4.86
Total Score	87.34
Relative Performance	97%

TABLE 6.0-9. ESAMS 2.7 Performance Analysis Worksheet.

Criterion	Performance Level			
	Poor	**Acceptable**	**Excellent**	**Score**
MOE #1 - Use of Standards:				
Criterion #1: Readability		33*4=132	67*5=335	467/100=4.67
Criterion #2: Modifiability		28*4=112	72*5=360	472/100=4.72
Criterion #3: ANSI standards		1*4=4	99*5=495	499/100=4.99
MOE #2 - Programming Conventions:				
Criterion #1: Use of comments and headers	10*3=30	15*4=60	75*5=375	465/100=4.65
Criterion #2: Use of formatted statements		6*4=24	94*5=470	494/100=4.94
Criterion #3: Logical I/O devices			100*5=500	500/100=5.00
Criterion #4: Variable declarations	7*3=21	6*4=24	87*5=435	480/100=4.80
Criterion #5: Variable initialization		11*4=44	89*5=445	489/100=4.89
Criterion #6: Variable naming conventions		10*4=40	90*5=450	490/100=4.90
Criterion #7: Algorithm clarity		5*4=20	95*5=475	495/100=4.95
MOE # 3 - Computational Efficiency:				
Criterion #1: Mixed mode calculations			100*5=500	500/100=5.00
Criterion #2: Use of library functions			100*5=500	500/100=5.00
Criterion #3: Nested computations		3*4=12	97*5=485	497/100=4.97
MOE # 4 - Maintainability:				
Criterion #1: Portability			100*5=500	500/100=5.00
Criterion #2: Memory management			100*5=500	500/100=5.00
Criterion #3: Use of COMMON blocks			100*5=500	500/100=5.00
Criterion #4: Modularity		14*4=56	86*5=430	486/100=4.86
Criterion #5: Subroutine traceability		100*4=400		400/100=4.00